Дата последней сборки документа:
5 декабря 2022 г. в 12:23

Конспект по книге Гудфеллоу «Глубокое обучение»*

Содержание

1. Численные методы

2. Основы машинного обучения

2.1. Точечная оценка

Точечное оценивание – это попытка найти единственное «наилучшее» предсказание интересующей величины. Пусть $\{x^{(1)}, \ldots, x^{(m)}\}$ – множество m независимых и одинаково распределенных точек. *Точечной оценкой*, или *статистикой*, называется любая функция этих данных

$$\theta_m = g(x^{(1)}, \ldots, x^{(m)}).$$

*Гудфеллоу Я., Бенджио И., Курвилль А. Глубокое обучение. – М.: ДМК Пресс, 2018. – 652 с.

В этом определении не требуется, чтобы g возвращала значение, близкое к истинному значению θ, ни даже чтобы область значений g совпадала со множеством допустимых значений θ.

Алгоритм k-групповой перекрестной проверки применяется для оценивания ошибки обобщения алгоритма обучения A, когда имеющийся набор данных $\mathbb{D}$ *слишком мал* для того, чтобы простое разделение на обучающий и тестовый или обучающий и контрольный наборы могло дать точную оценку ошибки обобщения, поскольку среднее значение потери L на малом тестовом наборе может иметь высокую дисперсию.

2.2. Смещение

Смещение оценки определяется следующим образом

$$\text{bias}(\hat{\theta}_m) = \mathbb{E}(\hat{\theta}_m) - \theta,$$

где математической ожидание вычисляется по данным (рассматривается как выборка из случайной величины), а θ – истинное значение параметра, которое определяет порождающее распределение.

Оценка $\hat{\theta}$ называется *несмещенной*, если

$$(\hat{\theta}_m) = 0, \text{ т.е. } \mathbb{E}(\hat{\theta}_m) = \theta.$$

Оценка $\hat{\theta}_m$ называется *асимптотически несмещенной*, если

$$\lim_{m\to\infty} \text{bias}(\hat{\theta}_m) = 0, \text{ т.е. } \lim_{m\to\infty} \mathbb{E}(\hat{\theta}_m) = \theta.$$

2.3. Дисперсия

Для определения смещения мы вычисляли математичесвкое ожидание оценки, но точно так же можем вычислить и ее дисперсию. *Дисперсией оценки* называется выражение

$$\text{Var}(\hat{\theta}).$$

Стандартной ошибкой $\text{SE}(\hat{\theta})$ называется квадратный корень из дисперсии.

Воспользовавшись центральной предельной теоремой, согласно которой среднее имеет приблизительно нормальное распределение, можем применить стандартную ошибку для вычисления вероятности того, что истинное математическое ожидание находится в выбранном интервале. Например, *95-процентный доверительный интервал* вокруг выборочного среднего (вокру оценки) $\hat{\mu}_m = \frac{1}{m}\sum_{k=1}^{n} x^{(i)}$ определяется формулой

$$(\hat{\mu}_m - 1.96\,\text{SE}(\hat{\mu}_m), \hat{\mu}_m + 1.96\,\text{SE}(\hat{\mu}_m))$$

при нормальном распределении со средним $\hat{\mu}_m$ и дисперсией $\text{SE}(\hat{\mu}_m)^2$.

NB: В экспериментах по машинному обучению принято говорить, что алгоритм A лучше алгоритма B, если верхняя граница 95-процентного доверительного интервала для ошибки алгоритма A меньше нижней границы 95-процентного доверительного интервала для ошибки алгоритма B.

2.4. Поиск компромисса между смещением и дисперсией для минимизации среднеквадратической ошибки

Что, если имеются две оценки, у одной из которых больше смещение, а у другой дисперсия? Какую выбрать?

Самый распространенный подход к выбору компромиссного решения – воспользоваться *перекрестной проверкой*. Эмпирически продемонстрировано, что перекрестная проверка дает отличные результаты во многих реальных задачах.

Можно также сравнить среднекваратическую ошибку (MSE) обеих оценок

$$\text{MSE} = \mathbb{E}[(\hat{\theta}_m - \theta)^2] = \text{bias}(\hat{\theta}_m)^2 + \text{Var}(\hat{\theta}_m)$$

Желательной является оценка с малой MSE, именно такие оценки держат под контролем и смещение, и дисперсию. Соотношение между смещением и дисперсией тесно связано с возникающими в машинном обучении понятиями емкости модели, недообучения и переобучения.

Если ошибка обобщения измеряется посредством MSE (и тогда смещение и дисперсия становятся важными компонентами ошибки обобщения), то увеличение емкости (то есть *усложение модели*) влечет за собой *повышение дисперсии* и *снижение смещения*.

2.5. Состоятельность

Обычно нас интересует также поведение оценки по мере роста размера обучающего набора. В частности, мы хотим, чтобы при увеличении числа примеров точечные оценки сходились к истинным значениям соответствующих параметров.

Формально это записывается в виде (условие состоятельности)

$$\hat{\theta}_m \xrightarrow{\mathbf{P}} \theta, \ (m \to \infty)$$

Иногда это условие называют *слабой состоятельностью*, понимая под *сильной состоятельностью* сходимость *почти наверное* $\hat{\theta}$ к θ.

Состоятельность гарантирует, что смещение оценки уменьшается с ростом числа примеров. Однако обратное неверно – из асимптотической несмещенности не вытекает состоятельность. Рассмотрим, к примеру, оценивание среднего μ нормального распределения $N(x; \mu, \sigma^2)$ по набору данных, содержащему m примеров: $\{x^{(1)}, \ldots, x^{(m)}\}$.

Можно было бы взять в качестве оценки первый пример: $\hat{\theta} = x^{(i)}$. В таком случае $\mathbb{E}(\hat{\theta})_m = \theta$, поэтому оценка является несмещенной вне зависимости от того, сколько примеров мы видели. Отсюда, конечно, следует, что оценка асимптотически несмещенная. Но она не является состоятельной, т.к. *неверно*, что $\hat{\theta}_m \to \theta, \ (m \to \infty)$.

2.6. Оценка максимального праводподобия

Рассмотрим множества m примеров $\mathbb{X} = \{x^{(1)}, \ldots, x^{(m)}\}$, независимо выбираемых из неизвестного порождающего распределения $p_{data}(x)$.

Обозначим $p_{model}(x; \theta)$ параметрическое семейство распределений вероятности над одним и тем же пространством, индексированное параметром θ.

Тогда оценка максимального правдоподобия для θ определяется формулой

$$\theta_{ML} = \arg\max_{\theta} p_{model}(\mathbb{X}; \theta) = \arg\max_{\theta} \prod_{i=1}^{m} p_{model}(x^{(i)}; \theta)$$

Такое произведение большого числа вероятностей по ряду причин может быть неудобно. Например, оно подвержено *потере значимости*. Для получения эквивалентной, но более удобной задачи оптимизации заметим, что взятие логарифма правдоподобия не изменяет arg max, но преобразует произведение в сумму

$$\theta_{ML} = \arg\max_{\theta} \sum_{i=1}^{m} \log p_{model}(x^{(i)}; \theta)$$

Поскольку arg max не изменяется при умножении функции стоимости на константу, мы можем разделить правую часть на m и получить выражение в виде математического ожидания относительно эмпирического распределения $\hat{p}_{data}$, определяемого обучающими данными

$$\theta_{ML} = \arg\max_{\theta} \mathbb{E}_{x \sim \hat{p}_{data}} [\log p_{model}(x; \theta)]$$

Один из способов интерпретации оценки максимального правдоподобия состоит в том, чтобы рассматривать ее как минимизацию дивергенции (расхождения) Кульбака-Лейблера между этими эмпирическим распределением $\hat{p}_{data}$, определяемым обучающим набором, и модельным распределением.

Дивергенция Кульбака-Лейблера определяется формулой

$$D_{KL}(\hat{p}_{data} \,||\, p_{model}) = \mathbb{E}_{x \sim \hat{p}_{data}} [\log \hat{p}_{data}(x) - \log p_{model}(x)]$$

Первый член разности в квадратных скобках зависит только от порождающего данные процесса, но не от модели. Следовательно, при обучении модели, минимизирующей дивергенцию КЛ, мы должны минимизировать только величину

$$-\mathbb{E}_{x \sim \hat{p}_{data}} [\log p_{model}(x)],$$

а это, конечно, то же самое, что максимизация величины $\theta_{ML} = \arg\max_{\theta} \mathbb{E}_{x \sim \hat{p}_{data}} [\log p_{model}(x; \theta)]$.

NB: То есть, другими словами задача максимизации правдоподобия эквивалентна задаче минимизации дивергенции Кульбака-Лейблера между эмпирическим распределением $\hat{p}_{data}$ и модельным распределением p_{model}.

2.7. Метод опорных векторов

Линейную функцию в методе опорных векторов можно переписать в виде

$$w^T x + b = b + \sum_{i=1}^{m} \alpha_i x^T x^{(i)},$$

где $x^{(i)}$ – обучающий пример, α – вектор коэффициентов.

Записав алгоритм обучения в таком виде, мы сможем заменить x результатом заданной функции признаков $\varphi(x)$, а скалярное произведение – функцией $k(x, x^{(i)}) = \varphi(x) \cdot \varphi(x^{(i)})$, которая называется ядром.

Заменив скалярное произведение вычислением ядра, мы можем делать предсказание, пользуясь функцией

$$f(x) = b + \sum_i \alpha_i k(x, x^{(i)})$$

Основанная на ядре функция в точности эквивалентна предварительной обработке путем применения $\varphi(x)$ ко всем входным данным с последующим обучением линейной модели в новом преобразованном пространстве.

NB: Трюк с ядром полезен по двум причинам:

- Во-первых, он позволяет обучать модели, *нелинейно* зависящие от x, применяя методы выпуклой оптимизации, о которых точно известно, что они сходятся эффективно
- Во-вторых, *ядерная функция* k часто допускает реализацию, значительно *более эффективную с вычислительной точки зрения*, чем наивное построение двух векторов $\varphi(x)$ и явное вычисление их скалярного произведения

Главный недостаток ядерных методов – тот факт, что сложность вычисления решающей функции линейно зависит от числа обучающих примеров, поскольку i-ый пример вносит член $\alpha_i k(x, x^{(i)})$ в решающую функцию.

В методе опорных векторов эта проблема сглаживается тем, что обучаемый вектор α содержит в основном нули. Тогда для классификации нового примера *требуется вычислить ядерную функцию только для обучающих примеров с ненулевыми* α_i. Эти обучающие примеры и называются опорными векторами.

2.8. Метод главных компонент

Метод главных компонент находит ортогональное линейное преобразование, переводящее входные данные x в представление z.

Рассмотрим матрицу плана X размера $m \times n$. Будем предполагать, что матемаческое ожидание данных $\mathbb{E}[x] = 0$. Если это не так, центрирования легко добиться, вычтя среднее из всех примеров на этапе предварительной обработки.

Несмещенная выборочная ковариационная матрица, ассоциированная с X, определяется по формуле

$$Var[x] = \frac{1}{m-1} X^T X$$

PCA находит представление (посредством линейного преобразования) $z = W^T x$, для которого $Var[z]$ – *диагональная.*

Главные компоненты можно получить с помощью сингулярного разложения. Точнее, это правые сингулярные верткоры. Чтобы убедиться в этом, предположим, что W – правые сингулярные векторы в разложении $X = U\Sigma W^T$. Тогда исходное уравнение собственных векторов можно переписать в базисе W

$$X^T X = (U\Sigma W^T)^T U\Sigma W^T = W\Sigma^2 W^T$$

Разложение SVD полезно для доказательства того, что PCA приводит к диагональной матрице $Var[z]$. Применяя сингулярное разложение X, мы можем выразить дисперсию X в виде

$$Var[x] = \frac{1}{m-1}X^TX = \frac{1}{m-1}(U\Sigma^2W^T)^TU\Sigma W^T = \frac{1}{m-1}W\Sigma^2W^T,$$

где используется тот факт, что $U^TU = I$, поскольку матрца U в сингулярном разложении по определению ортогональная. Отсюда следует, что ковариационная матрица z диагональная

$$Var[z] = \frac{1}{m-1}Z^TZ = \frac{1}{m-1}W^TX^TXW = \frac{1}{m-1}W^TW\Sigma^2W^TW = \frac{1}{m-1}\Sigma^2$$

На этот раз мы воспользовались тем, что $W^TW = I$ – опять же по определению сингулярного разложения.

Проведенный анализ показывает, что представление, полученное в результате проецирования данных x на z посредством линейного преобразования W, имеет *диагональную ковариационную матрицу* Σ^2. А отсюда сразу вытекает, что *взаимная корреляция отдельных элементов* z *равна нулю.*

2.9. Стохастический градиентный спуск

Стохастический градиентный спуск (СГС) имеет важные применения и за пределами глубокого обучения. Это основной способ обучения *больших линейных моделей* на очень больших наборах данных. Для модели фиксированноо размера стоимость одного шага СГС не зависит от размера обучающего набора m. Количество шагов до достижения сходимости обычно возрастает с ростом размера обучающего набора. Но когда m стремится к бесконечности, модель в итоге сходится к наилучшей возможной ошибке тестирования еще до того, как СГС проверил каждый пример из обучающего набора. Дальнейшее увеличение m не увеличивает время обуччения, необходимое для достижения наилучшей ошибки тестирования. С этой точки зрения можно сказать, что асимптотическая стоимость обучения модели методом СГС как функции от m имеет порядок $O(1)$.

2.10. Условное логарифмическое правдоподобие и среднеквадратическая ошибка

Линейую регрессию можно интерпретировать как *нахождение оценки максимального правдоподобия*. Будем считать, что цель не в том, чтобы вернуть одно предсказание $\hat{y}$, а чтобы построить модель, порождающую условное распределение $p(y|\mathbf{x})$. Цель алгоритма обучения теперь – аппроксимировать $p(y|\mathbf{x})$, подогнав его под все эти разные значения y, совместимые с $\mathbf{x}$.

Для вывода такого же алгоритма линейной регрессии, как и раньше, определим

$$p(y|\mathbf{x}) = N(y; \hat{y}(\mathbf{x}, \mathbf{w}), \sigma^2)$$

Функкция $\hat{y}(\mathbf{x}; \mathbf{w})$ дает предсказание среднего значения нормального распределения. В этом примере мы предполагаем, что дисперсия фиксирована и равна константе σ^2. Поскольку предполагается, что примеры независимы и одинаково распределены, то условное логарифмическое

правдоподобие записывается в виде

$$\sum_{i=1}^{m} \log p(y^{(i)}|\mathbf{x}^{(i)};\theta) = -m\,\log\sigma - \frac{m}{2}\log 2\pi - \sum_{i=1}^{m}\frac{\|\hat{y}^{(i)} - y^{(i)}\|^2}{2\sigma^2},$$

где $\hat{y}^{(i)}$ – результат линейной регрессии для i-ого примера $\mathbf{x}^{(i)}$, а m – число обучающих примеров.

Сравнивая логарифмическое правдоподобие со среднеквадратической ошибкой

$$\mathrm{MSE}_{train} = \frac{1}{m}\sum_{i=1}^{m}\|\hat{y}^{(i)} - y^{(i)}\|^2,$$

мы сразу же видим, что *максимизация логарифмического правдоподобия* относительно $\mathbf{w}$ дает ту же оценку параметров $\mathbf{w}$, что *минимизация среднеквадратической ошибки*.

Значения этих критериев различны, но положение оптимума совпадает. Это служит обоснованием использования среднеквадратической ошибки в качестве оценки максимального правдоподобия.

3. Глубокие сети прямого распространения

3.1. Обучение условных распределений с помощью максимального правдоподобия

Большинство современных нейронных сетей обучается *с помощью максимального правдоподобия*. Это означает, что в качестве функции стоимости берется отрицательное логарифмическое правдоподобие, которое можно эквивалентно описать как перекрестную энтропию между обучающими данными и распределением модели

$$J(\theta) = -\mathbb{E}_{x,y\sim\hat{p}_{\mathrm{data}}}\log p_{\mathrm{model}}(\mathbf{y}|\mathbf{x})$$

Одно необычное свойство перекрестной энтропии, используемой при вычислении оценки максимального правдоподобия, заключается в том, что для типичных встречающихся на практике моделей у нее, как правило, нет минимального значения. Если выходная величина дискретна, то в большинстве моделей параметризация устроена так, что модель неспособна представить вероятность 0 или 1, но может подойти к ней сколь угодно близко. Примером может служить логистическая регрессия.

3.2. Сигмоидные блоки и выходное распределение Бернулли

Во многих задачах требуется предсказывать значение бинарной величины y. В таком виде можно представить задачу классификации с двумя классами.

Подход на основе максимального правдоподобия заключается в определении распределения Бернулли величины y при условии $\mathbf{x}$.

Градиент **0** обычно приводит к проблемам, потому что у алгоритма обучения нет никаких указаний на то, как улучшить параметры.

Лучше применять другой подход, который гарантирует, что градиент обязательно будет достаточно большим, если модель дает неверный ответ. Этот подход основан на использовании сигмоидных выходных блоков в сочетании с максимальным правдоподобием.

Сигмоидный выходной блок

$$\hat{y} = \sigma(\mathbf{w}^T\mathbf{h} + b),$$

где σ – логистическая сигмоида.

3.3. Блоки softmax и категориальное выходное распределение

Если требуется представить распределение вероятности дискретной случайной величины, принимающей n значений, то можно воспользоваться функцией softmax. Ее можно рассматривать как обобщение сигмоиды, которая использовалась для представления распределения бинарной величины.

Функция softmax чаще всего используется как выход классификатора для представления распределения вероятности n классов. Реже функция softmax используется внутри самой модели.

Как и сигмоида, функция активации softmax склонна к насыщению. У сигмоиды всего один выход, и она насыщается, когда абсолютная величина аргумента очень велика. У softmax выходных значений несколько. Они насыщаются, когда велика абсолютная величина разностей между входными значениями.

softmax – это сглаженный вариант $\arg\max$. Сложность спектрального разложения матрицы $d \times d$ имеет порядок $O(d^3)$.

3.4. Скрытые блоки

Некоторые скрытые блоки, не являются всюду дифференцируемыми. Например, функция линейной ректификации (ReLU) $g(z) = \max\{0, z\}$ не дифференцируема в точке $z = 0$. Может показаться, что из-за этого g непригодна для работы с алгоритмами обучения градиентными методами. Но на практике градиентный спуск работает для таких моделей машинного обучения достаточно хорошо. Отчасти это связано с тем, что алгоритмы обучения нейронных сетей обычно не достигают локального минимума функции стоимости, а просто находят достаточно малое значение. Поскольку мы не ожидаем, что обучение выйдет на точку, где градиент равен 0, то можно смириться с тем, что минимум функции стоимости соответствует точкам, в которых градиент не определен. Недифференцируемые скрытые блоки обычно не дифференцируемы лишь в немногих точках. В общем случае функцию $g(z)$ имеет производную слева, определяемую коэффициентом наклона функции слева от z, и аналогично производную справа. Функция дифференцируема в точке z, только если производные слева и справа определены и равным между собой. Для функции $g(z) = \max\{0, z\}$ производная слева в точке $z = 0$ равна 0, а производная справа равна 1. В программных реализациях обучения нейронной сети обычно возвращается какая-то односторонняя производная, а не сообщается, что производная не определена и не возбуждается исключение. Эвристически это можно оправдать, заметив, что градиентная оптимизация на цифровом компьютере в любом случае подвержена численным погрешностям. Когда мы просим вычислить $g(0)$, крайне маловероятно, что истинное значение действительно равно 0. Скорее всего, это какое-то малое значение, округленное до 0.

Важно, что на практике можно *спокойно игнорировать недифференцируемость функции активации* скрытых блоков.

3.4.1. Блоки линейной ректификации и их обобщения

В блоке линейной ректификации используется функция активации $g(z) = \max\{0, z\}$. Эти блоки легко оптимизировать, потому что они очень похожи на линейные. Разница только в том, что блок линейной ректификации в половине своей области определения выводит 0. Поэтому производная блока линейной ректификации остается большой всюду, *где блок активен*.

Блоки линейной ректификации обычно применяются *после* аффинного преобразования

$$h = g(W^T x + b)$$

При инициализации параметров аффинного преобразования рекомендуется присваивать всем элементам b небольшое положительное значение, например, 0.1. Тогда блок линейной ректификации в начальный момент с большой вероятностью окажется активен для большинства обучающих примеров, и производная будет отлична от нуля.

Недостатком блоков линейной ректификации является невозможность обучить их градиентными методами на примерах, для которых функция активации блока равна нулю. Различные обобщения гарантируют, что градиент имеется в любой точке.

Три обобщения блоков линейной ректификации основаны на использовании ненулевого углового коэффициента α_i, когда $z_i < 0 : h_i = \max(0, z_i) + \alpha_i \min(0, z_i)$.

В случае абсолютной ректификации берутся фиксированные значения $\alpha_i = -1$, так что $g(z) = |z|$. Такая функция активации используется при распознавании объектов в изображении, где имеет смысл искать признаки, инвариантные относительно изменения полярности освещения. Другие обобщения находят более широкие применения. В случае ReLU с утечкой α_i принимаются равными фиксированному малому значению, например, 0.01, а в случае параметрического ReLU α_i считается обучаемым параметром.

Блоки линейной ректификации и все их обобщения основаны на принципе, согласно которому модель проще обучить, если ее поведение близко к линейному.

3.4.2. Логистическая сигмоида и гиперболический тангенс

Блоки линейной ректификации стали использоваться сравнительно недавно, а раньше большинство нейронных сетей в роли функции активации применялась логистическая сигмоида или гиперболический тангенс.

Сигмоидные блоки в качестве выходных предсказывают вероятность того, что бинарная величина равна 1. В отличие от кусочно-линейных, сигмоидные блоки близки к асимптоте в большей части своей области определения – приближаются к высокому значению, когда z стремится к бесконечности, и к низкому, когда z стремится к минус бесконечности. Высокой чувствительностью они обладают только в окрестности нуля. Из-за *насыщения* сигмоидальных блоков *градиентное обучение сильно затруднено*. Поэтому использование их в качестве скрытых блоков в сетях прямого распространения ныне не рекомендуется.

Если использовать сигмоидальную функцию активации необходимо, то лучше взять не логистическую сигмоиду, а гиперболический тангенс. Он ближе к тождественной функции в том смысле, что $\tanh(0) = 0$, тогда как $\sigma(0) = 1/2$.

Поскольку tanh походит на тождественную функцию в окрестности нуля, обучение глубокой нейронной сети $\hat{y} = w^T \tanh(U^T \tanh(V^T x))$ напоминает обучение линейной модели $\hat{y} = w^T U^T V^T x$,

при условии, что сигналы активации сети удается удерживать на низком уровне. При этом обучение сети с функцией активации tanh упрощается.

Сигмоидальные функции активации все же применяются, но не в сетях прямого распространения. К реккурнетным сетям, многим вероятностным моделям и некоторым автокодировщикам предъявляются дополнительные требования, исключающие использование кусочно-линейных функций.

3.5. Правило дифференцирования сложной функции

Это правило применяется для вычисления производных функций, являющихся композициями других функций, чьи производные известны.

Пусть $\mathbf{x} \in \mathbb{R}^m, \mathbf{y} \in \mathbb{R}^n$, g отображает $\mathbb{R}^m$ в $\mathbb{R}^n$, а f отображает $\mathbb{R}^n$ в $\mathbb{R}$. Тогда правило дифференцирования сложной функции в векторной форме запишется в виде

$$\nabla_x z = \left(\frac{\partial \mathbf{y}}{\partial \mathbf{x}}\right)^T \nabla_y z,$$

где $\partial \mathbf{y}/\partial \mathbf{x}$ – *матрица Якоби* функции g размера $n \times m$.

Отсюда видно, что градиент по переменной $\mathbf{x}$ можно получить, умножив матрицу Якоби $\partial \mathbf{y}/\partial \mathbf{x}$ на градиент $\nabla_y z$.

Обычно алгоритм обратного распространения применяется к тензорам произвольной размерности, а не просто к векторам. Концептуально это то же самое, что применение к векторам. Разница только в том, как числа организуются в сетке для представления тензора. Можно вообразить, что тензор сериализуется в вектор перед обратным распространением, затем вычисляется векторозначный градиент, после чего градиент снова преобразуется в тензор.

В таком переупорядоченном представлении обратное распространение – это все то же умножение якобиана на градиенты.

Для обозначения градиента значения z относительно тензора $\mathbf{X}$ мы пишем $\nabla_{\mathbf{X}} z$, как если бы $\mathbf{X}$ был просто вектором. Теперь индексы элементов $\mathbf{X}$ составные – например, трехмерный тензор индексируется тремя координатами. Мы можем абстрагироваться от этого различия, считая, что одна переменная i представляет целый кортеж индексов. Для любого возможного индексного кортежа i $(\nabla_{\mathbf{X}} z)_i$ обозначает частную производную $\partial z/\partial \mathbf{X}_i$ – точно так же, как для любого целого индекса i $(\nabla_x z)_i$ обозначает $\partial z/\partial x_i$. В этих обозначениях можно записать правило дифференцирования сложной функции в применении к тензорам. Если $\mathbf{Y} = g(\mathbf{X})$ и $z = f(\mathbf{Y})$, то

$$\nabla_{\mathbf{X}} z = \sum_j (\nabla_{\mathbf{X}} Y_j) \frac{\partial z}{\partial Y_j}$$

Алгоритм обратного распространения был разработан, чтобы избежать многократного вычисления одного и того же выражения при дифференцировании сложной функции. Из-за таких повторов время выполнения наивного алгоритма могло расти экспоненциально. Теперь, можно оценить вычислительную сложность алгоритма обратного распространения.

Если предположить, что стоимость вычисления всех операций приблизительно одинакова, то вычислительную сложность можно проанализировать в терминах количества выполненных операций. Следует помнить, что под единицей мы понимаем базовую единицу графа вычислений, в действительности она может состоять из нескольких арифметических операций (например, умножение матриц может считаться одной операцией в графе). Вычисление градиента в графе

с n вершинами никогда не приводит к выполнению или сохранению результатов более $O(n^2)$. Здесь мы подсчитываем в графе вычислений, а не отдельные аппаратные операции, поэтому важно понимать, что время выполнения разных операций может значительно различаться.

Легко видеть, что для вычисления градиента требуется не более $O(n^2)$ операций, потому что на этапе прямого распространения в худшем случае будут обсчитаны все n вершин исходного графа (в зависимости от того, какие значения мы хотим вычислить, может потребоваться обойти весь граф).

Поскольку граф вычислений – это ориентированный ациклический граф, число ребер в нем не более $O(n^2)$. Для типичных графов, встречающихся на практике, ситуация даже лучше. В большинстве нейронных сетей функции стоимости имеют в основном цепную структуру, так что сложность обратного распространения равна $O(n)$. Это намного лучше, чем наивный подход, при котором число обрабатываемых вершин иногда растет *экспоненциально*!

Откуда возникает экспоненциальный рост, можно понять, раскрыв и переписав правило дифференцирования сложной функции без рекурсии (здесь сумма по путям $u^{(\pi_1, u^{(\pi_2)}), \ldots, u^{(\pi_t)}}$ из $\pi_1 = j$ в $\pi_t = n$)

$$\frac{\partial u^{(n)}}{\partial u^{(j)}} = \sum \prod_{k=2}^{t} \frac{\partial u^{(\pi_k)}}{\partial u^{(\pi_{k-1})}}$$

Поскольку количество путей из вершины j в вершину n может экспоненциально зависит от длины пути, то число слагаемых в этой сумме, равное числу таких путей, может расти экспоненциально с увеличением глубины графа прямого распространения. Такая высокая сложность связана с многократным вычислением $\partial u^{(i)}/\partial u^{(j)}$. Чтобы *избежать повторных вычислений*, мы можем рассматривать *обратное распространение как алгоритм заполнения таблицы*, в которой храняться промежуточные результаты $\partial u^{(n)}/\partial u^{(i)}$.

Каждой вершине графа соответствует элемент таблицы, в котором хранится градиент для этой вершины. Заполняя таблицу в определенном порядке, алгоритм обратного распространения избегает повторного вычисления многих ошибок подвыражений. Такую стратегию заполнения таблицы иногда называют динамическим программированием.

Список литературы

1. *Рамальо Л.* Python – к вершинам мастерства: Лаконичное и эффективное программирование. – М.: МК Пресс, 2022. – 898 с.
2. *Хейдт М., Груздев А.* Изучаем pandas. – М.: ДМК Пресс, 2019. – 682 с.